

Information integration services, Part 1: Service technologies for information integration

By: Kevin N Wang and Beate Porst

Introduction

Information Services is one of the core parts of the IBM SOA Reference Architecture, and are required for managing and accessing all forms of information in a Service Oriented Architecture. Thereby, Information Services emphasize on separating the physical implementation or location of the information from the processes and application that request the information.

The idea behind Information Services is to provide and access information through industry standards which then allows building business processes without the requirement to understand data source specific interfaces or the semantic and complexity of the functionality that these services deliver.

Figure 1 is IBM's SOA Reference Architecture, which shows the key capabilities that are required for comprehensive, enterprise wide SOA solutions. Information Services is a key component in the reference architecture as it provides the capabilities to federate, replicate and transform data sources that might be implemented in a variety of ways.

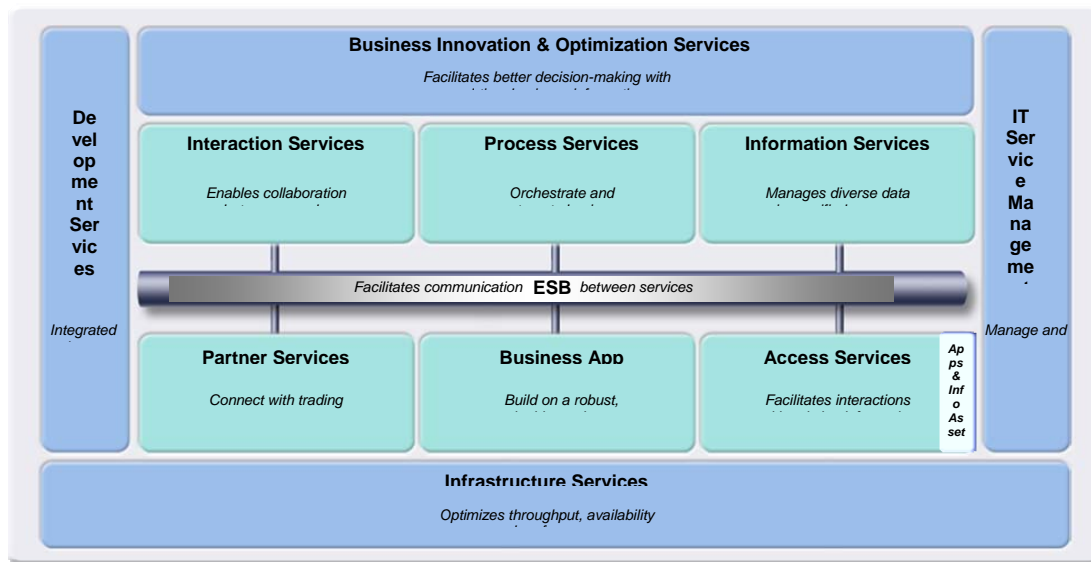


Figure 1. Information services in the IBM SOA Reference Architecture

By definition, there is no particular technology in which Information Services have to be developed as long as four main principles are involved:

- interoperability
- modularity
- reusability
- componentization

However, Web service technology takes on an increasing significance as the interface in which Information Services are provided. But Information Services have specific characteristics for messages sizes, invocation pattern and service runtime. Therefore, it is necessary to understand these technologies to make the right choice for SOA implementation.

In this series of three whitepapers we will provide a comparison of Web services technologies, followed by a study of different Web services types for integration specific problems, provided both by IBM WebSphere SOA Edition and IBM Rational Application Developer. Finally we will outline an SOA case study, which will use Web service technology for Information Services, and at the same time, indicate that Information Services are only components in an overall SOA architecture.

Generally, Information Services comprises the complete spectrum of Information Management functionality. However, for this series we have narrowed the focus to Information Integration Services, which is a specific type of service that provides real-time, integrated access to business information regardless of data location or format.

Evaluation of Service Technologies (EJBs and Web Services)

Services Oriented Architecture (SOA) is a design pattern for distributed software systems that promotes the decomposition of business functions into a series of self contained, services. Thereby, SOA neither imposes any specific technology in which these services have to be implemented nor how these services have to be assembled or choreographed. As Enterprise Architects face the choice for the “right” service technology, we have tried to shed some light on this by evaluating some of the available services technology.

Our selection of service technologies was driven by two factors:

- The service technology is typically used for Information Integration Services
- Loose coupling. Here, with the exception of EJB we have only chosen loosely coupled service types

Enterprise Java Beans

Enterprise Java Beans (EJB) is the server side components of Java Platform, Enterprise Edition (Java EE, formerly known as J2EE) for developing and running distributed Java based applications (please see the Resources link for more information). A new specification for EJB (EJB 3.0) was just finalized in May 2006 which is supposed to

simplify EJB development through the use of annotations. However, at the time of writing this article, most tools and application servers still only supported EJB 2.0.

EJBs are similar to ordinary Java classes, but must provide specific interfaces for container and client access. In addition, they can only run in an EJB container that manages the EJB life cycle behavior.

EJB components come in three varieties, each with its own defined role:

- **Session beans** may be either stateful or stateless and are primarily used to encapsulate business logic, carry out tasks on behalf of a client, and act as controllers or managers for other beans.
- **Entity beans** represent persistent objects or business concepts that exist beyond a specific application's lifetime. Entity beans can be developed using bean-managed persistence (BMP), implemented by the developer or container-managed persistence (CMP), or implemented by the container.
- **Message-driven beans** listen asynchronously for Java Message Service (JMS) messages from any client or component and are used for loosely coupled, typically batch-type, processing.

EJBs can be used to build a Service Oriented Architecture by componentizing business logic in Entity beans and encapsulating them with stateless session beans. Remote procedure calls are made possible by Java Remote Method Invocation (RMI), where distributed beans communicate with each other through their remote interfaces. The transport layer for RMI could be either the proprietary Java Remote Method Protocol (JRMP) or the Internet Inter-ORB protocol (IIOP) from CORBA (RMI-IIOP).

Web services

There is no common formal definition for the term Web service but the two major consortiums that are working on Web services specifications define them following the following manner:

w3 Consortium: Web Services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.

OASIS Consortium: A Web Service is a software component that is described via WSDL (Web Services Description Language) and is capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP.

In a Service Oriented Architecture built with Web services, applications on different machines interact with each other through published interfaces described in Web Services Description Language (WSDL) and exchange data with standardized, SOAP and XML based messaging. Services can be registered and discovered on a service registry such as UDDI-based registries.

In this article, except where explicitly qualified, the term “Web services” refers to the style of Web services characterized by the use of the Simple Object Access Protocol (SOAP) running on the HTTP transport.

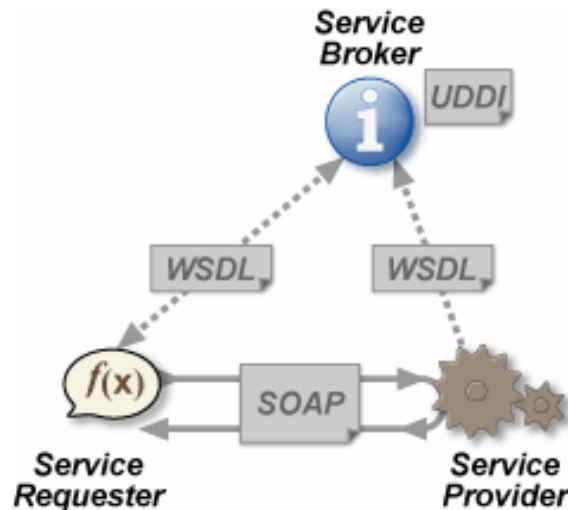


Figure 2. Web services architecture (source: http://en.wikipedia.org/wiki/Web_service)

EJB vs. Web services

The main advantage of an EJB solution over a Web service solution is better performance and enterprise level robustness, including security, reliability and transaction support, all of which are required for vendors by the Java EE specification.

The main advantage of Web services is their excellent interoperability among diverse systems and platforms.

Below is a comparison of the two non-functional system characteristics

- **Extensibility and interoperability** – Since Web services are based on open and widely accepted standards (such as SOAP, XML and HTTP), they enjoy good interoperability among platforms and vendor products. This interoperability also gives Web services-based SOA systems good future extensibility and integration capability with other third party applications. EJB solutions interoperate well within the Java world, but are less interoperable with applications not residing on the Java platform (such as .NET).
- **Heterogeneous integration:**

- Web services are platform and implementation agnostic. They can bring together applications on different systems (Mainframes, UNIX, LINUX, Windows and others) and they can be implemented in different programming languages (for example, Java, C++, Python and Visual Basic).
- EJBs, even though they are available on most operating systems usually form a more homogeneous solution where both server and client side are implemented in Java. However, using IIOP also makes it possible invoke an RMI application from any other programming language compliant with CORBA.
- **Coupling:** Coupling defines the degree of dependencies among services or systems. Thereby, loose coupling is a key attribute of SOA solutions as it allows for the following:
 - Quick assembly of business solutions by combining services from a variety of systems.
 - The ability to make changes to a component without affecting others
 - Reusing services independently as neither the consumer nor the provider of the service have to know each other.

Web service interfaces are basically “just” wrappers, defining in an abstract, canonical data format independent of any programming language or operating system. They are, in this sense, loose-coupled, independent and self-contained components. EJB+RMI components are coupled on both the programming language (Java) and the environment of the Java platform.

- **Descriptiveness:** Web services use Web Services Description Language (WSDL) to describe exposed interfaces and messages. WSDL is XML based and are both human understandable and supported widely by software tools. EJB doesn't have a similar service description mechanism similar to WSDL.
- **Reliability:**
 - The weak point of system reliability for Web services-based systems is the transport layer, because the HTTP protocol does not guarantee delivery of messages. The WS-Reliability specification is intended to address this problem. However, as this is a relatively new specification, not many Web services provider applications support the specification.
 - EJB+RMI, when running on top of IIOP provides transport reliability.
- **Security:**
 - Web services can obtain transport layer security if it is used with SSL (Secured Socket Layer), which gives authentication and secure delivery of messages. The WS-Security standard, when it is incorporated, has support for message level security, which has better integration with other Web services standards, but it also brings extra performance overhead. Several upcoming standards (now in draft status) are aiming to provide Web services more

security features such as authorization, policy, roles and secure conversation sessions.

- Besides transport level security, the Java EE platform mandates conformance with the CSIV2 standard (Common Secure Interoperability, Version 2) that enables interoperable authentication, delegation and privileges. Most vendors also support functions such as identity assertion, single sign-on, user registry and integration with third party security providers, although they are not part of the Java EE security specification. The Java platform doesn't provide message level security.
- **Performance:**
 - Web services are self-described text-based messages, meaning each message carries its own metadata. In average a Web service contains about 50% metadata. "Verboseness" negatively effects performance, however, the right choice of SOAP engine (SAX based engines generally perform better then DOM) and encoding style (Document-literal encoding performs better then RPC) can make a substantial difference for the overall web service performance
 - In comparison, EJBs use binary messages and do not carry additional meta data overhead, which consequently give them better performance.
- **Transaction support:**
 - Transactions are a fundamental concept in building reliable distributed applications. A transaction is a mechanism to insure all the participants in an application achieve a mutually agreed outcome. For Web Services, the transaction concept is not part of the Web services specifications (WSDL, SOAP). But SOA solutions increasingly tie together large number of participants to form distributed applications. To provide consistency for these applications has led to the specification for WS-Transaction and WS-Coordination. Current support for WS- Transaction is mostly triggered by the application server capability (on both sides, client and provider).
 - Java EE has transaction specification (JTA and JTS) that vendors implement to provide transaction support.

Decision points in the information integration services context:

- **Existing investment in technologies:** In an Information Integration Services context, the decision to expose the services via EJB+RMI/IIOP or Web services first depends on the architecture of the system where such services are consumed. If existing infrastructure is Java EE based, using the same technology to bring in information integration services gives better performance and system maintainability. On the other hand, if existing infrastructure is a heterogeneous solution, Web services is a better choice.

- **Desired level of fine-grained control and business logic association with the data to be serviced:** If the services and the back end data are tightly coupled with business logic (for example, access roles and life cycle control), it is possible to wrap such services in EJBs and have the business logic programmed in. If the goal is to surface data and information services via a more standardized way, Web services is preferred.
- **Tools SOA support and development skills:** Many types of Information integration services can be built using off-the-shelf products. (For example, IBM WebSphere DataStage is to build data transformation services.) If such products come with SOA support that can generate either Web Services or EJBs automatically (in case of WebSphere SOA Edition, it can generate both), it will greatly simplify the integration effort. Otherwise, programmatically exposing back end services via EJB+RMI involves heavier coding and potentially longer development cycle and deeper technical skills. Web services could be a faster and more cost-effective way with the availability of tools that automatically generate Web services from back end functions (for example, IBM Rational Application Developer can generate Web services from Java applications)
- **Desired non-functional system characteristics listed above:** In general, EJB+RMI is more suitable to build mission critical systems/subsystems for the high performance, reliability and security requirements; Web services is a better choice when future extensibility, system flexibility and interoperability with software from other parties are major considerations.

SOAP-based and RESTful Web services – an architectural style choice

SOAP-based Web Services

Since the introduction of the concept of Web services, the SOAP (Simple Object Access Protocol) protocol has played an important role to form the foundation of the Web services protocol stack. It is an XML-based messaging protocol for exchanging information in a distributed environment.

Not only does SOAP provides a messaging format that communicating parties agree and operate on, it also serves as the layer that other higher level protocols converge and interoperate, thanks to the extensibility of the SOAP protocol. Other high level standards, such as WS-Security, WS-Reliability, WS-Transaction, WS-Policy, etc. can plug themselves in by annotating the SOAP header with information about security, transaction context or session state. Together, they form an extensible framework that provides the syntax and semantics that Web services operate on. We call Web services with this type of architectural style “SOAP-based Web services”.

Advantages of SOAP-based Web services

- SOAP is the only message binding specified in the WS-I Basic Profile. The WS-I Basic Profile is a specification put out by the Web Services Interoperability (WS-I) organization, which is an open industry organization chartered to promote Web service interoperability.
- The SOAP structure is designed to be extensible, so SOAP-based Web services can easily integrate with WS-* or other future related standards as building blocks to gain enterprise class semantics.
- SOAP has good development tooling support from vendors.
- SOAP can run on multiple transport protocols through bindings (most commonly HTTP, also JMS, SMTP, etc)

Drawbacks of SOAP-based Web services

- SOAP messages carry extra metadata so it is more verbose, which leads to performance overhead. The new SOAP 1.2 alleviates this by allowing compression.
- With the addition of higher level standards such as OASIS' WS-*, SOAP-based Web services can become complex and hard to manage.

RESTful Web services

REST stands for Representational State Transfer. It is an emerging alternative architectural style for building Web services. The concept was first brought up by Roy Fielding in his doctoral dissertation “Architectural Styles and the Design of Network-based Software Architecture” (please see the Resources section for more information) in 2000. REST follows the same architectural design philosophy of the already widely successful World Wide Web, aiming to provide Web services the same scalability and flexibility, and interoperability.

REST uses the standards-based protocols of HTTP and XML for transport and messages. To interact with a RESTful Web service, an application issues a HTTP request to an URI and one of GET/POST/PUT/DELETE to achieve the equivalent of CRUD operations (Create, Retrieve, Update and Delete) against the resource.

In summary, the design philosophies for REST are:

- Universal resource identification: all system elements are treated as “resources” with unique URIs.

- Stateless messaging: every REST messages are self contained with all the information necessary to understand and consume it. This means that it is not necessary for either the server or the client to maintain application context.
- Interlinked resources: REST resources contain URIs linking to representations of other resources. It is possible to navigate from representation of a resource to representation of other resources.
- A set of predefined operations: Four HTTP operations are used to access and alter states of resources: GET – retrieve; POST – update ; PUT – create; DELETE - destroy

Advantages of RESTful Web services:

- RESTful Web services achieves interoperability with vendors and applications by using only HTTP and XML, both of which are highly supported protocols.
- RESTful Web services is loosely coupled and flexible. As long as resources maintain the same URIs to the outside, they can evolve overtime independently.
- Simple interfaces and standard operations make programming against such Web services very easy.
- RESTful Web services perform well in comparison to SOAP. As an example, according to Amazon, “querying Amazon using REST is 6 times faster than with SOAP”. (please see the Resources section for more information)
- Basic Security (message integrity, authentication, authorization) is pushed down to network protocols and Web servers.

Disadvantages of RESTful Web services

- REST doesn't have a framework to provide enterprise class semantics such as transactions and advanced security. Such characteristics are left to developers to implement.
- RESTful Web services inherit HTTP's limitations in asynchrony and reliable messaging. And since HTTP is the only choice as transport protocol for REST, those limitations cannot be overcome by simply replacing HTTP with another transport protocol. They have to be implemented by developers as well.

SOAP-based or RESTful – comparison and decision points

The goals of SOAP-based Web services and RESTful Web services are the same – to provide a common ground that distributed programs interoperate on. Instead of inventing new protocols to achieve this, they both take the approach of building on the already widely accepted Internet protocols of XML and HTTP*.

The fundamental difference between the two is more of **what** they expose and interconnect than how they connect them. SOAP-based Web Services expose methods

and operations that manipulate backend data records and RESTful Web Services expose data records themselves with standardized operations (CRUD).

SOAP-based Web services in general are more closely aligned with an enterprise architecture and therefore target:

- Intra-enterprise environment where data access and manipulation is controlled and process driven;
- Higher level system semantics are of priority, such as advanced security, roles and policies, governance.
- Integrating legacy applications that interface through APIs

RESTful Web services are more “Web 2.0 ’ish”, and suited for

- Services exposed to outside of the company boundary to the Internet often as building blocks for other applications or “mash-ups”
- Providing simple services with low skill barrier for consumption to gain adoption
- Deployment in a fast changing environment that requires components of the system to constantly evolve to adapt.

In an Information Integration Services context, since both the tasks (such as, data transformation and data cleansing) and their application environment are of enterprise type in nature, most of the services are implemented as SOAP-based Web services. However, it is possible to use the REST style to build services with simple semantics (such as, to start or stop a predefined service).

JMS – An alternative transport for Web services

Where HTTP doesn't suffice

Conceptually, Web services can leverage different transport protocols, including HTTP, JMS and SMTP. However, the majority of Web services are provided for HTTP. There are two reasons to that:

1. The current WSDL specification only defines endpoint definitions for HTTP
2. Compatibility and interoperability.

Nonetheless, HTTP protocol has its inherent limitations with enterprise class requirements – mostly its lack of support for asynchronous message delivery, and other enterprise semantics such as security and reliability. The WS-* protocols address some of the concerns, but only to add more performance overhead.

Introducing JMS

JMS (Java Messaging Services) is a messaging API that defines the standard for J2EE applications specification to “create, send, receive, and read messages. It enables

distributed communication that is loosely coupled, reliable, and asynchronous.” [\[link\]](#)
JMS provides Web services a message driven transport with enterprise strength.

Advantages of JMS:

- JMS supports synchronous and asynchronous messaging patterns. With the asynchronous messaging pattern, message sender doesn't have to wait for message receiver's response to complete the task. This decouples the availability dependency between system modules and makes communication more flexible. Asynchronous invocation is particular useful for long-running services which otherwise would block a service client for a long time.
- JMS guarantees message delivery via the once and only once delivery semantics of the persistent messages.
- JMS allows interleaving of requests, so that a communication channel can be shared by multiple requesters through multiplexing. This helps with system performance scalability when bandwidth is the restricting resource.
- JMS supports local transactions by grouping operations into an atomic work unit, and rolls back if any of the operation fails. The JMS standard itself doesn't require distributed transaction support, but many vendors support this via Java Transaction API (JTA).

Drawbacks of JMS:

- WSDL 1.2 does not specify a notation for JMS as binding by itself nor how to specify a JMS endpoint address.
- JMS transport requires message sender and receiver both run on the Java platform. This restricts its reach or requires mediation technology to connect to non-Java clients.
- Different vendor implementations of JMS are slightly different. Interoperability between different JMS systems could be an issue.

Decision points: HTTP or JMS

- **Interaction patterns:** If the interaction patterns between components are mostly synchronous request-respond, HTTP transport is a good choice. On the other hand, JMS simplifies the asynchronous interaction pattern by providing an asynchronous message delivery mechanism.
- **Interoperability requirements and future extensibility:** Generally, if services are going to be integrated in very heterogeneous systems would demand to provide HTTP endpoint. Also, any service intended to cross the enterprise boundary should be provided through HTTP.
- **Enterprise level semantics requirements:** JMS is a more robust enterprise level transport with many built in support for enterprise requirements such as reliability messaging and transactions, and gives better system performance.

In the Information Integration Services context, the JMS transport is favored when there are requirements for reliability and asynchrony of either the flow of data or the invocations of surfaced functions or jobs.

Summary

In this article, we introduced EJBs and Web services as service technologies to implement SOA systems and made comparison in the context of Information Integration Services. We further discussed different architectural styles and transports for Web services and made recommendations on some key decision points. In real service engagements, architects have to take into consideration of many factors such as exiting information system architecture, desired non-functional characteristics and available resources to make the decision.

Resources

1. Enterprise Java Beans: <http://java.sun.com/products/ejb/>
2. RESTful Web services:
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
3. RESTful Web services performance compared to SOAP based Web services according to Amazon: (fourth paragraph)
http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html

About the authors

Kevin N. Wang is a Technical Solution Specialist working for the Information Integration Technology Solutions team in IBM Software group. His current area of interests includes Information Management products and solutions in a SOA context. Kevin lives in Mountain View, CA and can often be found in nearby coffee shops and bookstores.

Beate Porst is a member of the IIS Product Strategy team. She has an extensive background in software development and customer support for Information Management products. In her previous role as Solution Architect in the SOA solutions team, Beate was working with customers and business partners to adopt SOA and IBM's Information as Service mission. Beate holds a Master of Computer Science degree from the University of Rostock, Germany. Her interests include Data Federation, Web Services, XML and DB2.